

# Categorical Phase Semantics for Linear Logic

---

Riley Shahar  
Steve Zdancewic





- Categorical semantics, an introduction

- Categorical semantics, an introduction
- Linear logic and phase semantics

- Categorical semantics, an introduction
- Linear logic and phase semantics
- Our work: categorical phase semantics

- Categorical semantics, an introduction
- Linear logic and phase semantics
- Our work: categorical phase semantics
- Conclusion & future work

*PLs*

*Logics*

*Categories*

$PLs \xrightarrow{\text{Curry-Howard}} Logics$

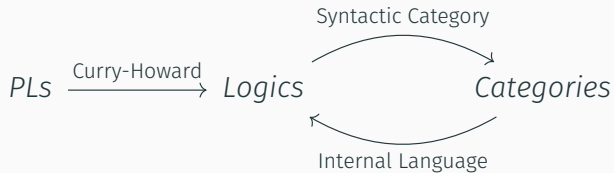
Categories



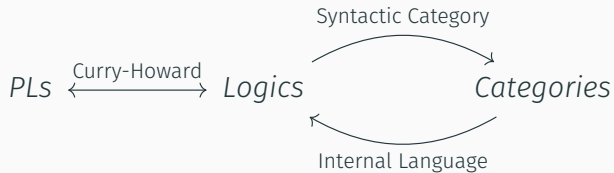
# The Story



# The Story



# The Story





We have some *atomic propositions*

$X, Y, Z, \dots$

# Signatures

We have some *atomic propositions*

$X, Y, Z, \dots$

some *propositional connectives*

$\neg, \wedge, \vee, \rightarrow, T, F, \dots$

# Signatures

We have some *atomic propositions*

$$X, Y, Z, \dots$$

some *propositional connectives*

$$\neg, \wedge, \vee, \rightarrow, T, F, \dots$$

These form a *signature (language, syntax)*,

$$\mathcal{L} = \{X, \neg Y, (X \wedge Y) \rightarrow (T \vee X), \dots\}.$$





# Inference Rules

We define some *inference rules* on our signature:

$$\frac{A \quad B}{A \wedge B} \wedge I \qquad \frac{A \quad A \rightarrow B}{B} \rightarrow E.$$

# Inference Rules

We define some *inference rules* on our signature:

$$\frac{A \quad B}{A \wedge B} \wedge I \qquad \frac{A \quad A \rightarrow B}{B} \rightarrow E.$$

We can use these to write *proof trees*:

$$\frac{\frac{A \quad A \rightarrow B}{B} \rightarrow E \quad B \rightarrow C}{C} \rightarrow E.$$

The inference rules determine\* a *entailment* relation  $\vdash$  on  $\mathcal{L}$ :

$A \vdash B$  whenever  $B$  is provable from  $A$ .

The inference rules determine\* a *entailment* relation  $\vdash$  on  $\mathcal{L}$ :

$A \vdash B$  whenever  $B$  is provable from  $A$ .

The *identity axiom* asserts

$$\frac{}{A \vdash A} \text{Id.}$$

The inference rules determine\* a *entailment* relation  $\vdash$  on  $\mathcal{L}$ :

$A \vdash B$  whenever  $B$  is provable from  $A$ .

The *identity axiom* asserts

$$\frac{}{A \vdash A} \text{ID.}$$

The *cut rule* asserts

$$\frac{A \vdash B \quad B \vdash C}{A \vdash C} \text{CUT.}$$

# Provability Semantics

The inference rules determine\* a *entailment* relation  $\vdash$  on  $\mathcal{L}$ :

$A \vdash B$  whenever  $B$  is provable from  $A$ .

The *identity axiom* asserts

$$\frac{}{A \vdash A} \text{ID.}$$

The *cut rule* asserts

$$\frac{A \vdash B \quad B \vdash C}{A \vdash C} \text{CUT.}$$

Any logic with these rules has an associated preorder, its *provability semantics*.

## Provability Semantics of (Classical) Conjunction

# Provability Semantics of (Classical) Conjunction

What do we know about  $A \wedge B$ ?



# Provability Semantics of (Classical) Conjunction

What do we know about  $A \wedge B$ ?

$$A \wedge B \vdash A \quad A \wedge B \vdash B$$

# Provability Semantics of (Classical) Conjunction

What do we know about  $A \wedge B$ ?

$$A \wedge B \vdash A \quad A \wedge B \vdash B$$

$$\frac{Z \vdash A \quad Z \vdash B}{Z \vdash A \wedge B}$$

# Provability Semantics of (Classical) Conjunction

What do we know about  $A \wedge B$ ?

$$A \wedge B \vdash A \quad A \wedge B \vdash B$$

$$\frac{Z \vdash A \quad Z \vdash B}{Z \vdash A \wedge B}$$

So, conjunction is just the meet (infimum, glb).

# Provability Semantics of (Classical) Conjunction

What do we know about  $A \wedge B$ ?

$$A \wedge B \vdash A \quad A \wedge B \vdash B$$

$$\frac{Z \vdash A \quad Z \vdash B}{Z \vdash A \wedge B}$$

So, conjunction is just the meet (infimum, glb).

You can keep going: you'll get a *Heyting algebra* or a *Boolean algebra*.

We can talk about *proof objects*:

# Proof Objects

We can talk about *proof objects*:

$x : A$  means  $x$  is a proof of  $A$ .

# Proof Objects

We can talk about *proof objects*:

$x : A$  means  $x$  is a proof of  $A$ .

$\Gamma \vdash x : A$  means  $x$  is a proof of  $A$  under some assumptions  $\Gamma$ .

# Proof Objects

We can talk about *proof objects*:

$x : A$  means  $x$  is a proof of  $A$ .

$\Gamma \vdash x : A$  means  $x$  is a proof of  $A$  under some assumptions  $\Gamma$ .

Alternate notation:

$$\begin{array}{c} \vdots \\ \vdots \\ \vdots \end{array} x \quad \begin{array}{c} \Gamma \\ \vdots \\ \vdots \\ \vdots \end{array} x$$

$A \quad , \quad A$



# The Curry-Howard Isomorphism

# The Curry-Howard Isomorphism

$$\frac{\frac{\begin{array}{c} \vdots \\ \vdots \\ x \end{array} \quad \begin{array}{c} \vdots \\ \vdots \\ y \end{array}}{A \quad B}}{A \wedge B} \\ \hline A$$

# The Curry-Howard Isomorphism

$$\frac{\frac{\begin{array}{c} \vdots \\ x \end{array} \quad \begin{array}{c} \vdots \\ y \end{array}}{A \quad B}}{A \wedge B} \\ \hline A$$

```
fn (x:A, y:B) -> A {  
  p = (x, y);  
  return fst p;  
}
```

# The Curry-Howard Isomorphism

$$\frac{\frac{\begin{array}{c} \vdots \\ \vdots \end{array} x \quad \begin{array}{c} \vdots \\ \vdots \end{array} y}{A \quad B}}{A \wedge B} \\ \hline A$$

```
fn (x:A, y:B) -> A {  
  p = (x, y);  
  return fst p;  
}
```

These are both just  $x$ , and for the *same reason*.

# The Curry-Howard Isomorphism

$$\frac{\frac{\begin{array}{c} \vdots \\ x \end{array} \quad \begin{array}{c} \vdots \\ y \end{array}}{A \quad B}}{A \wedge B} \\ \hline A$$

```
fn (x:A, y:B) -> A {  
  p = (x, y);  
  return fst p;  
}
```

These are both just  $x$ , and for the *same reason*.

This is the *Curry-Howard isomorphism*:

*Types are propositions; programs are proofs.*



The *identity axiom* asserts

$$\frac{}{x : A \vdash x : A} \text{Id.}$$

The *identity axiom* asserts

$$\frac{}{x : A \vdash x : A} \text{ID.}$$

The *cut rule* asserts

$$\frac{\Gamma \vdash x : A \quad A \vdash y : B}{\Gamma \vdash \text{cut}(x, y) : B} \text{CUT.}$$



# Proof Semantics

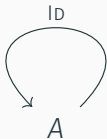
The *identity axiom* asserts

$$\frac{}{x : A \vdash x : A} \text{ID.}$$

The *cut rule* asserts

$$\frac{\Gamma \vdash x : A \quad A \vdash y : B}{\Gamma \vdash \text{cut}(x, y) : B} \text{CUT.}$$

Put another way,



# Proof Semantics

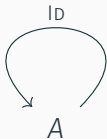
The *identity axiom* asserts

$$\frac{}{x : A \vdash x : A} \text{ID.}$$

The *cut rule* asserts

$$\frac{\Gamma \vdash x : A \quad A \vdash y : B}{\Gamma \vdash \text{cut}(x, y) : B} \text{CUT.}$$

Put another way,



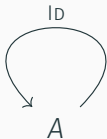
$$\Gamma \xrightarrow{x} A \xrightarrow{y} B.$$

# Proof Semantics

The *identity axiom* asserts

$$\frac{}{x : A \vdash x : A} \text{ID.}$$

Put another way,



The *cut rule* asserts

$$\frac{\Gamma \vdash x : A \quad A \vdash y : B}{\Gamma \vdash \text{cut}(x, y) : B} \text{CUT.}$$



# Proof Semantics

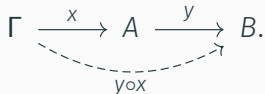
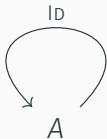
The *identity axiom* asserts

$$\frac{}{x : A \vdash x : A} \text{ID.}$$

The *cut rule* asserts

$$\frac{\Gamma \vdash x : A \quad A \vdash y : B}{\Gamma \vdash \text{cut}(x, y) : B} \text{CUT.}$$

Put another way,



These is a *category*! Any logic (or language) has its *syntactic category* as its *proof semantics*.



# Categories

$A$

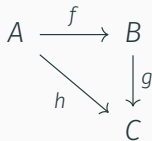
$B$

$C$

A *category* consists of:

- Objects

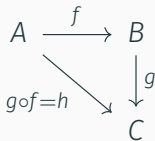
# Categories



A *category* consists of:

- Objects
- Morphisms

# Categories

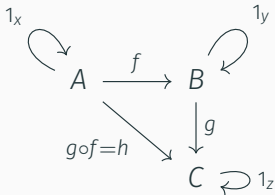


A *category* consists of:

- Objects
- Morphisms
- Composition



# Categories



A *category* consists of:

- Objects
- Morphisms
- Composition
- Identities

# Proof Semantics of (Classical) Conjunction

# Proof Semantics of (Classical) Conjunction

Again: what do we know about  $A \wedge B$ ?

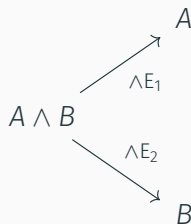
$A$

$A \wedge B$

$B$

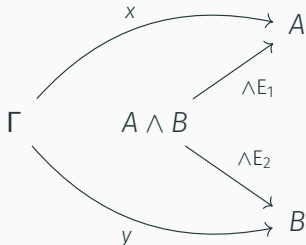
# Proof Semantics of (Classical) Conjunction

Again: what do we know about  $A \wedge B$ ?



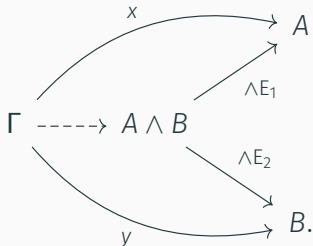
# Proof Semantics of (Classical) Conjunction

Again: what do we know about  $A \wedge B$ ?



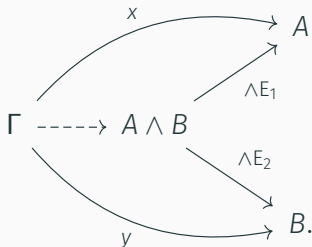
# Proof Semantics of (Classical) Conjunction

Again: what do we know about  $A \wedge B$ ?



# Proof Semantics of (Classical) Conjunction

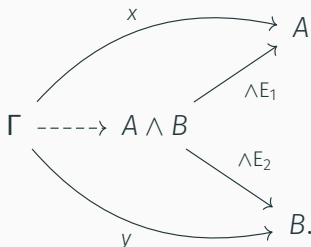
Again: what do we know about  $A \wedge B$ ?



This is the *categorical product*.

# Proof Semantics of (Classical) Conjunction

Again: what do we know about  $A \wedge B$ ?



This is the *categorical product*.

Again, you can keep going: you'll get (something like) a *catesian closed category*.



# Why can we do math with sets?

## Why can we do math with sets?

We ground all of math in sets. But, sets are just some category, of sets and set-functions.

# Why can we do math with sets?

We ground all of math in sets. But, sets are just some category, of sets and set-functions.

The categorical explanation is that sets form a *topos*.

# Why can we do math with sets?

We ground all of math in sets. But, sets are just some category, of sets and set-functions.

The categorical explanation is that sets form a *topos*.

One important property is that *subsets are predicates*:

# Why can we do math with sets?

We ground all of math in sets. But, sets are just some category, of sets and set-functions.

The categorical explanation is that sets form a *topos*.

One important property is that *subsets are predicates*:

$$A \subseteq X \rightarrow (x \mapsto x \in A)$$

# Why can we do math with sets?

We ground all of math in sets. But, sets are just some category, of sets and set-functions.

The categorical explanation is that sets form a *topos*.

One important property is that *subsets are predicates*:

$$\begin{aligned} A \subseteq X &\rightarrow (x \mapsto x \in A) \\ (P : X \rightarrow \{0, 1\}) &\rightarrow \{x \in X : P(x) = 1\}. \end{aligned}$$

# Why can we do math with sets?

We ground all of math in sets. But, sets are just some category, of sets and set-functions.

The categorical explanation is that sets form a *topos*.

One important property is that *subsets are predicates*:

$$\begin{aligned} A \subseteq X &\rightarrow (x \mapsto x \in A) \\ (P : X \rightarrow \{0, 1\}) &\rightarrow \{x \in X : P(x) = 1\}. \end{aligned}$$

# Why can we do math with sets?

We ground all of math in sets. But, sets are just some category, of sets and set-functions.

The categorical explanation is that sets form a *topos*.

One important property is that *subsets are predicates*:

$$\begin{aligned} A \subseteq X &\rightarrow (x \mapsto x \in A) \\ (P : X \rightarrow \{0, 1\}) &\rightarrow \{x \in X : P(x) = 1\}. \end{aligned}$$

How can we frame this categorically?





# Subobjects

Subsets have *inclusion functions*

$$i: A \rightarrow X$$

$$a \mapsto a$$

which are *injective*.

# Subobjects

Subsets have *inclusion functions*

$$i: A \rightarrow X$$

$$a \mapsto a$$

which are *injective*.

Fix a class  $\mathcal{M}$  of morphisms, which includes the identities and is closed under composition. A *subobject*  $A$  of an object  $X$  is a morphism

$$i: A \hookrightarrow X.$$

in  $\mathcal{M}$ .

# Subobjects

Subsets have *inclusion functions*

$$i: A \rightarrow X$$

$$a \mapsto a$$

which are *injective*.

Fix a class  $\mathcal{M}$  of morphisms, which includes the identities and is closed under composition. A *subobject*  $A$  of an object  $X$  is a morphism

$$i: A \hookrightarrow X.$$

in  $\mathcal{M}$ .

If you pick  $\mathcal{M}$  to be *monomorphisms*, you get the normal notion of subsets, subgroups, subspaces, etc.



Let  $\text{Sub}_{\mathcal{C}}(X)$  denote the subobjects of an object  $X$  of  $\mathcal{C}$ .

# Predicate Semantics

Let  $\text{Sub}_{\mathcal{C}}(X)$  denote the subobjects of an object  $X$  of  $\mathcal{C}$ .

$\text{Sub}_{\mathcal{C}}(X)$  is naturally ordered by compatible inclusion:

# Predicate Semantics

Let  $\text{Sub}_{\mathcal{C}}(X)$  denote the subobjects of an object  $X$  of  $\mathcal{C}$ .

$\text{Sub}_{\mathcal{C}}(X)$  is naturally ordered by compatible inclusion:

$$i \leq j \quad \text{iff}$$



# Predicate Semantics

Let  $\text{Sub}_{\mathcal{C}}(X)$  denote the subobjects of an object  $X$  of  $\mathcal{C}$ .

$\text{Sub}_{\mathcal{C}}(X)$  is naturally ordered by compatible inclusion:

$$i \leq j \quad \text{iff} \quad \begin{array}{ccc} A & \xhookrightarrow{i} & X \\ & & \uparrow j \\ & & B. \end{array}$$

# Predicate Semantics

Let  $\text{Sub}_{\mathcal{C}}(X)$  denote the subobjects of an object  $X$  of  $\mathcal{C}$ .

$\text{Sub}_{\mathcal{C}}(X)$  is naturally ordered by compatible inclusion:

$$i \leq j \quad \text{iff} \quad \begin{array}{ccc} A & \xrightarrow{i} & X \\ & \searrow \exists! & \uparrow j \\ & & B. \end{array}$$

# Predicate Semantics

Let  $\text{Sub}_{\mathcal{C}}(X)$  denote the subobjects of an object  $X$  of  $\mathcal{C}$ .

$\text{Sub}_{\mathcal{C}}(X)$  is naturally ordered by compatible inclusion:

$$i \leq j \quad \text{iff} \quad \begin{array}{ccc} A & \xrightarrow{i} & X \\ & \searrow \exists! & \uparrow j \\ & & B. \end{array}$$

# Predicate Semantics

Let  $\text{Sub}_{\mathcal{C}}(X)$  denote the subobjects of an object  $X$  of  $\mathcal{C}$ .

$\text{Sub}_{\mathcal{C}}(X)$  is naturally ordered by compatible inclusion:

$$i \leq j \quad \text{iff} \quad \begin{array}{ccc} A & \xrightarrow{i} & X \\ & \searrow \exists! & \uparrow j \\ & & B. \end{array}$$

This is a preorder, since  $\mathcal{M}$  is closed under composition and has identities. Hence, subobjects give provability semantics for a logic.

# Predicate Semantics

Let  $\text{Sub}_{\mathcal{C}}(X)$  denote the subobjects of an object  $X$  of  $\mathcal{C}$ .

$\text{Sub}_{\mathcal{C}}(X)$  is naturally ordered by compatible inclusion:

$$i \leq j \quad \text{iff} \quad \begin{array}{ccc} A & \xrightarrow{i} & X \\ & \searrow \exists! & \uparrow j \\ & & B. \end{array}$$

This is a preorder, since  $\mathcal{M}$  is closed under composition and has identities. Hence, subobjects give provability semantics for a logic.

With sets, the power set is a Heyting (Boolean) algebra. Such categories are called *Heyting (Boolean) categories*. These categories form the *predicate semantics* of intuitionistic (classical) logic.



# Refinement Types

Consider this program:

```
fn divide(a: int, b: nat) -> ratl {  
  return a / b;  
}
```

# Refinement Types

Consider this program:

```
fn divide(a: int, b: nat) -> ratl {  
  return a / b;  
}
```

Is this safe?



# Refinement Types

Consider this program:

```
fn divide(a: int, b: nat) -> ratl {  
  return a / b;  
}
```

Is this safe? No! Division by zero!

# Refinement Types

Consider this program:

```
fn divide(a: int, b: nat) -> ratl {  
  return a / b;  
}
```

Is this safe? No! Division by zero!

We want to write

```
fn divide(a: int, b: (nat such that b > 0)) -> ratl {  
  return a / b;  
}
```

# Refinement Types

Consider this program:

```
fn divide(a: int, b: nat) -> ratl {  
  return a / b;  
}
```

Is this safe? No! Division by zero!

We want to write

```
fn divide(a: int, b: (nat such that b > 0)) -> ratl {  
  return a / b;  
}
```

This is a predicate! So we probably want our type system to be a Heyting category. Category theory can tell us what we need to do to get that.



Due to Girard (1987).

Due to Girard (1987).

Provides a *logic of limited resources*.

Due to Girard (1987).

Provides a *logic of limited resources*.

**Classical Logic**

Due to Girard (1987).

Provides a *logic of limited resources*.

## Classical Logic

$\text{fn } x \ y \Rightarrow (x, y)$  ✓



Due to Girard (1987).

Provides a *logic of limited resources*.

## Classical Logic

$\text{fn } x \ y \Rightarrow (x, y)$  ✓

$\text{fn } x \Rightarrow (x, x)$  ✓

Due to Girard (1987).

Provides a *logic of limited resources*.

## Classical Logic

`fn x y => (x, y)` ✓

`fn x => (x, x)` ✓

`fn f (x, y) => f x y` ✓

Due to Girard (1987).

Provides a *logic of limited resources*.

## Classical Logic

$\text{fn } x \ y \Rightarrow (x, y)$  ✓

$\text{fn } x \Rightarrow (x, x)$  ✓

$\text{fn } f \ (x, y) \Rightarrow f \ x \ y$  ✓

$\text{fn } (x, y) \Rightarrow x$  ✓

$\text{fn } (x, y) \Rightarrow y$  ✓

Due to Girard (1987).

Provides a *logic of limited resources*.

## Classical Logic

`fn x y => (x, y)` ✓

`fn x => (x, x)` ✓

`fn f (x, y) => f x y` ✓

`fn (x, y) => x` ✓

`fn (x, y) => y` ✓

## Linear Logic

# Linear Logic

Due to Girard (1987).

Provides a *logic of limited resources*.

## Classical Logic

$\text{fn } x \ y \Rightarrow (x, y)$  ✓

$\text{fn } x \Rightarrow (x, x)$  ✓

$\text{fn } f \ (x, y) \Rightarrow f \ x \ y$  ✓

$\text{fn } (x, y) \Rightarrow x$  ✓

$\text{fn } (x, y) \Rightarrow y$  ✓

## Linear Logic

$\text{fn } x \ y \multimap x \otimes y$  ✓

# Linear Logic

Due to Girard (1987).

Provides a *logic of limited resources*.

## Classical Logic

$\text{fn } x \ y \Rightarrow (x, y)$  ✓

$\text{fn } x \Rightarrow (x, x)$  ✓

$\text{fn } f \ (x, y) \Rightarrow f \ x \ y$  ✓

$\text{fn } (x, y) \Rightarrow x$  ✓

$\text{fn } (x, y) \Rightarrow y$  ✓

## Linear Logic

$\text{fn } x \ y \multimap x \otimes y$  ✓

$\text{fn } x \multimap x \otimes x$  ✗

# Linear Logic

Due to Girard (1987).

Provides a *logic of limited resources*.

## Classical Logic

$\text{fn } x \ y \Rightarrow (x, y)$  ✓

$\text{fn } x \Rightarrow (x, x)$  ✓

$\text{fn } f \ (x, y) \Rightarrow f \ x \ y$  ✓

$\text{fn } (x, y) \Rightarrow x$  ✓

$\text{fn } (x, y) \Rightarrow y$  ✓

## Linear Logic

$\text{fn } x \ y \multimap x \otimes y$  ✓

$\text{fn } x \multimap x \otimes x$  ✗

$\text{fn } f \ (x \otimes y) \multimap f \ x \ y$  ✓

# Linear Logic

Due to Girard (1987).

Provides a *logic of limited resources*.

## Classical Logic

$\text{fn } x \ y \Rightarrow (x, y)$  ✓

$\text{fn } x \Rightarrow (x, x)$  ✓

$\text{fn } f \ (x, y) \Rightarrow f \ x \ y$  ✓

$\text{fn } (x, y) \Rightarrow x$  ✓

$\text{fn } (x, y) \Rightarrow y$  ✓

## Linear Logic

$\text{fn } x \ y \multimap x \otimes y$  ✓

$\text{fn } x \multimap x \otimes x$  ✗

$\text{fn } f \ (x \otimes y) \multimap f \ x \ y$  ✓

$\text{fn } (x \otimes y) \multimap x$  ✗

$\text{fn } (x \otimes y) \multimap y$  ✗





Also due to Girard.

# Phase Semantics

Also due to Girard.

An early provability semantics for linear logic.

# Phase Semantics

Also due to Girard.

An early provability semantics for linear logic.

Works with any *classical monoid*  $M$ .

# Phase Semantics

Also due to Girard.

An early provability semantics for linear logic.

Works with any *classical monoid*  $M$ .

$$\llbracket A \rrbracket \subseteq M$$

# Phase Semantics

Also due to Girard.

An early provability semantics for linear logic.

Works with any *classical monoid*  $M$ .

$$\begin{aligned}\llbracket A \rrbracket &\subseteq M \\ \llbracket A \otimes B \rrbracket &= \text{cl}(\llbracket A \rrbracket \llbracket B \rrbracket) \\ &= \text{cl}\{ab : a \in \llbracket A \rrbracket, b \in \llbracket B \rrbracket\}\end{aligned}$$

# Phase Semantics

Also due to Girard.

An early provability semantics for linear logic.

Works with any *classical monoid*  $M$ .

$$\llbracket A \rrbracket \subseteq M$$

$$\begin{aligned}\llbracket A \otimes B \rrbracket &= \text{cl}(\llbracket A \rrbracket \llbracket B \rrbracket) \\ &= \text{cl}\{ab : a \in \llbracket A \rrbracket, b \in \llbracket B \rrbracket\}\end{aligned}$$

$$\llbracket A \multimap B \rrbracket = \{c \in M : \forall a \in \llbracket A \rrbracket, ac \subseteq \llbracket B \rrbracket\}$$

# Phase Semantics

Also due to Girard.

An early provability semantics for linear logic.

Works with any *classical monoid*  $M$ .

$$\llbracket A \rrbracket \subseteq M$$

$$\begin{aligned}\llbracket A \otimes B \rrbracket &= \text{cl}(\llbracket A \rrbracket \llbracket B \rrbracket) \\ &= \text{cl}\{ab : a \in \llbracket A \rrbracket, b \in \llbracket B \rrbracket\}\end{aligned}$$

$$\begin{aligned}\llbracket A \multimap B \rrbracket &= \{c \in M : \forall a \in \llbracket A \rrbracket, ac \subseteq \llbracket B \rrbracket\} \\ &= \bigcup \{C \subseteq M : \llbracket A \rrbracket C \subseteq \llbracket B \rrbracket\}\end{aligned}$$



## Example Applications of Phase Semantics

Technical lemmas about linear logic [Girard 1987, Okada 1999].

## Example Applications of Phase Semantics

Technical lemmas about linear logic [Girard 1987, Okada 1999].

Undecidability of boolean bunched implications [LG 2010].

## Example Applications of Phase Semantics

Technical lemmas about linear logic [Girard 1987, Okada 1999].

Undecidability of boolean bunched implications [LG 2010].

Safety of concurrent constraint programs [FRS 2001, HB 2008].

## Example Applications of Phase Semantics

Technical lemmas about linear logic [Girard 1987, Okada 1999].

Undecidability of boolean bunched implications [LG 2010].

Safety of concurrent constraint programs [FRS 2001, HB 2008].

Takeaway: phase semantics are *tunable*.



$$\llbracket A \otimes B \rrbracket = \text{cl}(\llbracket A \rrbracket \llbracket B \rrbracket); \quad \llbracket A \multimap B \rrbracket = \bigcup \{C \subseteq M : \llbracket A \rrbracket C \subseteq \llbracket B \rrbracket\}$$

$$\llbracket A \otimes B \rrbracket = \text{cl}(\llbracket A \rrbracket \llbracket B \rrbracket); \quad \llbracket A \multimap B \rrbracket = \bigcup \{C \subseteq M : \llbracket A \rrbracket C \subseteq \llbracket B \rrbracket\}$$

Yetter (1990) observed that you don't need elements to do phase semantics.



$$\llbracket A \otimes B \rrbracket = \text{cl}(\llbracket A \rrbracket \llbracket B \rrbracket); \quad \llbracket A \multimap B \rrbracket = \bigcup \{C \subseteq M : \llbracket A \rrbracket C \subseteq \llbracket B \rrbracket\}$$

Yetter (1990) observed that you don't need elements to do phase semantics.

*Quantales* are preorders with a multiplication and joins (sups, lubs) which distribute over multiplication. Examples include the power set of any monoid.

$$\llbracket A \otimes B \rrbracket = \text{cl}(\llbracket A \rrbracket \llbracket B \rrbracket); \quad \llbracket A \multimap B \rrbracket = \bigcup \{C \subseteq M : \llbracket A \rrbracket C \subseteq \llbracket B \rrbracket\}$$

Yetter (1990) observed that you don't need elements to do phase semantics.

*Quantales* are preorders with a multiplication and joins (sups, lubs) which distribute over multiplication. Examples include the power set of any monoid.

*Quantale semantics* are the provability semantics of linear logic.

Linear logic has categorical semantics in *closed symmetric monoidal categories*.

Linear logic has categorical semantics in *closed symmetric monoidal categories*.

Intuitionistic Logic

Linear logic has categorical semantics in *closed symmetric monoidal categories*.

Intuitionistic Logic

---

Semantics	Structure
-----------	-----------

---

Provability	Heyting Algebras
-------------	------------------

# Categorical Semantics of Linear Logic

Linear logic has categorical semantics in *closed symmetric monoidal categories*.

Intuitionistic Logic

Semantics	Structure
Provability	Heyting Algebras
Proof	CCCs

# Categorical Semantics of Linear Logic

Linear logic has categorical semantics in *closed symmetric monoidal categories*.

## Intuitionistic Logic

Semantics	Structure
Provability	Heyting Algebras
Proof	CCCs
Predicate	Heyting Categories

# Categorical Semantics of Linear Logic

Linear logic has categorical semantics in *closed symmetric monoidal categories*.

Intuitionistic Logic

Linear Logic

Semantics	Structure
Provability	Heyting Algebras
Proof	CCCs
Predicate	Heyting Categories



# Categorical Semantics of Linear Logic

Linear logic has categorical semantics in *closed symmetric monoidal categories*.

## Intuitionistic Logic

Semantics	Structure
Provability	Heyting Algebras
Proof	CCCs
Predicate	Heyting Categories

## Linear Logic

Semantics	Structure
Provability	Quantales

# Categorical Semantics of Linear Logic

Linear logic has categorical semantics in *closed symmetric monoidal categories*.

## Intuitionistic Logic

Semantics	Structure
Provability	Heyting Algebras
Proof	CCCs
Predicate	Heyting Categories

## Linear Logic

Semantics	Structure
Provability	Quantales
Proof	CSMCs

# Categorical Semantics of Linear Logic

Linear logic has categorical semantics in *closed symmetric monoidal categories*.

## Intuitionistic Logic

Semantics	Structure
Provability	Heyting Algebras
Proof	CCCs
Predicate	Heyting Categories

## Linear Logic

Semantics	Structure
Provability	Quantales
Proof	CSMCs
Predicate	??

# Our Contribution

We construct a predicate semantics of linear logic, capturing phase semantics as a special case.

We construct a predicate semantics of linear logic, capturing phase semantics as a special case.

- We need a multiplicative structure on subobjects. We can get this via *monoid objects*.

We construct a predicate semantics of linear logic, capturing phase semantics as a special case.

- We need a multiplicative structure on subobjects. We can get this via *monoid objects*.
- We need joins on subobjects. We can get this via *coproducts*.

We construct a predicate semantics of linear logic, capturing phase semantics as a special case.

- We need a multiplicative structure on subobjects. We can get this via *monoid objects*.
- We need joins on subobjects. We can get this via *coproducts*.

To make subobjects closed under these operations, we need to take *images*.

## A Worked Example



## A Worked Example

Define

$$\mathbf{List}X := \{\text{lists with elements in } X\}$$

## A Worked Example

Define

$\text{List}X := \{\text{lists with elements in } X\}$

$\text{Maybe}X := X \sqcup \{\perp\}$

## A Worked Example

Define

$$\mathbf{List}X := \{\text{lists with elements in } X\}$$
$$\mathbf{Maybe}X := X \sqcup \{\perp\}$$
$$\mathbf{List}_{2\mathbb{N}}X := \{l \in \mathbf{List}X : 2 \mid \text{len}(l)\}$$

## A Worked Example

Define

$$\mathbf{List}X := \{\text{lists with elements in } X\}$$
$$\mathbf{Maybe}X := X \sqcup \{\perp\}$$
$$\mathbf{List}_{2\mathbb{N}}X := \{l \in \mathbf{List}X : 2 \mid \text{len}(l)\}$$

Let's compute  $\mathbf{List}_{2\mathbb{N}} \otimes \mathbf{Maybe}$ .

## A Worked Example

Define

$$\mathbf{List}X := \{\text{lists with elements in } X\}$$
$$\mathbf{Maybe}X := X \sqcup \{\perp\}$$
$$\mathbf{List}_{2\mathbb{N}}X := \{l \in \mathbf{List}X : 2 \mid \text{len}(l)\}$$

Let's compute  $\mathbf{List}_{2\mathbb{N}} \otimes \mathbf{Maybe}$ . Let  $X = \{1, 2\}$ .

## A Worked Example

Define

$$\mathbf{List}X := \{\text{lists with elements in } X\}$$

$$\mathbf{Maybe}X := X \sqcup \{\perp\}$$

$$\mathbf{List}_{2\mathbb{N}}X := \{l \in \mathbf{List}X : 2 \mid \text{len}(l)\}$$

Let's compute  $\mathbf{List}_{2\mathbb{N}} \otimes \mathbf{Maybe}$ . Let  $X = \{1, 2\}$ . Compute  $\mathbf{List}_{2\mathbb{N}} \circ \mathbf{Maybe}X$

## A Worked Example

Define

$$\mathbf{List}X := \{\text{lists with elements in } X\}$$

$$\mathbf{Maybe}X := X \sqcup \{\perp\}$$

$$\mathbf{List}_{2\mathbb{N}}X := \{l \in \mathbf{List}X : 2 \mid \text{len}(l)\}$$

Let's compute  $\mathbf{List}_{2\mathbb{N}} \otimes \mathbf{Maybe}$ . Let  $X = \{1, 2\}$ . Compute

$$\mathbf{List}_{2\mathbb{N}} \circ \mathbf{Maybe}X$$

$$= \{ [], [1, 1], [1, \perp], [\perp, 2], [\perp, \perp], \dots \}$$

## A Worked Example

Define

$$\mathbf{List}X := \{\text{lists with elements in } X\}$$

$$\mathbf{Maybe}X := X \sqcup \{\perp\}$$

$$\mathbf{List}_{2\mathbb{N}}X := \{l \in \mathbf{List}X : 2 \mid \text{len}(l)\}$$

Let's compute  $\mathbf{List}_{2\mathbb{N}} \otimes \mathbf{Maybe}$ . Let  $X = \{1, 2\}$ . Compute

$$\mathbf{List}_{2\mathbb{N}} \circ \mathbf{Maybe}X$$

$$= \{ \quad [], \quad [1, 1], \quad [1, \perp], \quad [\perp, 2], \quad [\perp, \perp], \quad \dots \}$$

$$\simeq \{ \quad [], \quad [[1], [1]], \quad [[1], []], \quad [ [], [2]], \quad [ [], []], \quad \dots \} \quad \text{by including}$$



## A Worked Example

Define

$$\mathbf{List}X := \{\text{lists with elements in } X\}$$

$$\mathbf{Maybe}X := X \sqcup \{\perp\}$$

$$\mathbf{List}_{2\mathbb{N}}X := \{l \in \mathbf{List}X : 2 \mid \text{len}(l)\}$$

Let's compute  $\mathbf{List}_{2\mathbb{N}} \otimes \mathbf{Maybe}$ . Let  $X = \{1, 2\}$ . Compute

$$\mathbf{List}_{2\mathbb{N}} \circ \mathbf{Maybe}X$$

$$= \{ \quad [], \quad [1, 1], \quad [1, \perp], \quad [\perp, 2], \quad [\perp, \perp], \quad \dots \}$$

$$\simeq \{ \quad [], \quad [[1], [1]], \quad [[1], []], \quad [ [], [2]], \quad [ [], []], \quad \dots \} \quad \text{by including}$$

$$\rightsquigarrow \{ \quad [], \quad [1, 1], \quad [1], \quad [2], \quad [], \quad \dots \} \quad \text{by concating}$$

## A Worked Example

Define

$$\mathbf{List}X := \{\text{lists with elements in } X\}$$

$$\mathbf{Maybe}X := X \sqcup \{\perp\}$$

$$\mathbf{List}_{2\mathbb{N}}X := \{l \in \mathbf{List}X : 2 \mid \text{len}(l)\}$$

Let's compute  $\mathbf{List}_{2\mathbb{N}} \otimes \mathbf{Maybe}$ . Let  $X = \{1, 2\}$ . Compute

$$\mathbf{List}_{2\mathbb{N}} \circ \mathbf{Maybe}X$$

$$= \{ \quad [], \quad [1, 1], \quad [1, \perp], \quad [\perp, 2], \quad [\perp, \perp], \quad \dots \}$$

$$\simeq \{ \quad [], \quad [[1], [1]], \quad [[1], []], \quad [ [], [2]], \quad [ [], []], \quad \dots \} \quad \text{by including}$$

$$\rightsquigarrow \{ \quad [], \quad [1, 1], \quad [1], \quad [2], \quad [], \quad \dots \} \quad \text{by concating}$$

$$\rightsquigarrow \{ \quad [], \quad [1, 1], \quad [1], \quad [2], \quad \dots \} \quad \text{by quotienting.}$$

## A Worked Example

Define

$$\mathbf{List}X := \{\text{lists with elements in } X\}$$

$$\mathbf{Maybe}X := X \sqcup \{\perp\}$$

$$\mathbf{List}_{2\mathbb{N}}X := \{l \in \mathbf{List}X : 2 \mid \text{len}(l)\}$$

Let's compute  $\mathbf{List}_{2\mathbb{N}} \otimes \mathbf{Maybe}$ . Let  $X = \{1, 2\}$ . Compute

$$\mathbf{List}_{2\mathbb{N}} \circ \mathbf{Maybe}X$$

$$= \{ \quad [], \quad [1, 1], \quad [1, \perp], \quad [\perp, 2], \quad [\perp, \perp], \quad \dots \}$$

$$\simeq \{ \quad [], \quad [[1], [1]], \quad [[1], []], \quad [ [], [2]], \quad [ [], []], \quad \dots \} \quad \text{by including}$$

$$\rightsquigarrow \{ \quad [], \quad [1, 1], \quad [1], \quad [2], \quad [], \quad \dots \} \quad \text{by concating}$$

$$\rightsquigarrow \{ \quad [], \quad [1, 1], \quad [1], \quad [2], \quad \dots \} \quad \text{by quotienting.}$$

Generalizing,  $\mathbf{List}_{2\mathbb{N}} \otimes \mathbf{Maybe} = \mathbf{List}$ .



Study these semantics in specific settings

Study these semantics in specific settings

- Giry monad

Study these semantics in specific settings

- Giry monad
- Linear refinement types

Study these semantics in specific settings

- Giry monad
- Linear refinement types

Study completeness and decidability via these semantics



Study these semantics in specific settings

- Giry monad
- Linear refinement types

Study completeness and decidability via these semantics

Completely characterize predicate semantics for linear logic

# Summary

## Summary

Our construction turns subobject posets of monoid objects in certain categories into models of linear logic: a form of *predicate semantics for linear logic*.

# Summary

Our construction turns subobject posets of monoid objects in certain categories into models of linear logic: a form of *predicate semantics for linear logic*.

Special cases give a linearization of monads, as in the previous example; Girard's phase semantics; and nondeterministic phase semantics.

# Summary

Our construction turns subobject posets of monoid objects in certain categories into models of linear logic: a form of *predicate semantics for linear logic*.

Special cases give a linearization of monads, as in the previous example; Girard's phase semantics; and nondeterministic phase semantics.

Concretely, these semantics give a useful tool for analyzing type systems which depend on linear predicates for proof.

# Summary

Our construction turns subobject posets of monoid objects in certain categories into models of linear logic: a form of *predicate semantics for linear logic*.

Special cases give a linearization of monads, as in the previous example; Girard's phase semantics; and nondeterministic phase semantics.

Concretely, these semantics give a useful tool for analyzing type systems which depend on linear predicates for proof.

Thanks! :)